ELSEVIER

Theoretical Computer Science 295 (2003) 189–203

Theoretical Computer Science

www.elsevier.com/locate/tcs

Converting two-way nondeterministic unary automata into simpler automata

Viliam Geffert[a,1], Carlo Mereghetti[b], Giovanni Pighizzini[c,*]

[a] *Department of Computer Science, P.J. Šafárik University, Jesenná 5, 04154 Košice, Slovakia*

[b] *Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano, Bicocca, via Bicocca degli Arcimboldi 8, 20126 Milano, Italy*

[c] *Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39, I-20135 Milano, Italy*

Received 22 August 2001; received in revised form 29 October 2001; accepted 8 January 2002

Abstract

We show that, on inputs of length exceeding $5n^2$, any n-state unary two-way nondeterministic finite automaton (2nfa) can be simulated by a $(2n+2)$-state quasi-sweeping 2nfa. Such a result, besides providing a "normal form" for 2nfa's, enables us to get a subexponential simulation of unary 2nfa's by two-way deterministic finite automata (2dfa's). In fact, we prove that any n-state unary 2nfa can be simulated by a sweeping 2dfa with $\mathcal{O}(n^{\lceil \log_2(n+1)+3 \rceil})$ states.

Keywords: Formal languages; Finite state automata; Unary languages

1. Introduction

The problem of evaluating the costs—in terms of states—of the simulations between different kinds of finite state automata has been widely investigated in the literature. In particular, several contributions deal with *two-way* automaton simulations (see, e.g., [3,10,11,13]).

The main open question in this field is certainly that posed by Sakoda and Sipser in 1978 [11], which asks for *the cost of turning a two-way nondeterministic or a*

* Corresponding author.

E-mail addresses: geffert@kosice.upjs.sk (V. Geffert), mereghetti@disco.unimib.it (C. Mereghetti), pighizzi@dsi.unimi.it (G. Pighizzini).

[1] Supported by the Slovak Grant Agency for Science (VEGA) under contract #1/7465/20 "Combinatorial Structures and Complexity of Algorithms."

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00403-6

one-way nondeterministic n-state finite state automaton (*resp.* $2nfa, 1nfa$) *into a two-way deterministic finite state automaton* ($2dfa$). They conjecture such a cost to be exponential, and Sipser [12] proves that this is exactly the case when 2dfa's are required to be *sweeping automata* (*qsdfa*), i.e., 2dfa's having head reversals *only* at the ends of the input tape. Indeed, from a descriptional point of view, qsdfa's turn out to be weaker than general 2dfa's. In fact, in [1,9], some families of languages are exhibited for which 2dfa's are actually exponentially more succinct than qsdfa's. However, sweeping mode represents a meaningful and natural simplification to gain interesting partial answers and ways of tackling the general question.

Berman and Lingas [2] state a lower bound of $\Omega(n^2/\log n)$ for cost of 2nfa's vs. 2dfa's simulation, and provide an interesting connection with the celebrated open problem DLOGSPACE $\stackrel{?}{=}$ NLOGSPACE. More precisely, they show that if DLOGSPACE = NLOGSPACE then, for some polynomial p, and for any integer m and k-state 2nfa A, there is a $p(mk)$-state 2dfa accepting a subset of $L(A)$, the language accepted by A. The subset consists of all strings of length not exceeding m in $L(A)$. As a consequence of this result, Sipser [12] relates the DLOGSPACE $\stackrel{?}{=}$ NLOGSPACE question also to the existence of qsdfa's with a polynomial number of states for a certain family of regular languages. This might give additional evidence that the problem of evaluating the costs of two-way automaton simulations is not only motivated by the investigation on the succinctness of representing regular languages but is also related to fundamental questions in complexity. A further improvement is contained in [4], where the lower bound for the cost of 2nfa's vs. 2dfa's simulation is raised to $\Omega(n^2)$.

One of the most promising restrictions of the open question of Sakoda–Sipser is represented by its *unary version* which leads us to study optimal simulations between *unary automata*, i.e., automata working with a single letter input alphabet. The problem of evaluating the costs of unary automata simulations was first settled in [12] and has lead to emphasize some relevant differences with the general case. For instance, we know that $\mathcal{O}(e^{\sqrt{n \ln n}})$ states suffice to simulate a unary n-state 1nfa or 2dfa by a one-way deterministic finite state automaton (1dfa). Furthermore, a unary n-state 1nfa can be simulated by a 2dfa having $\mathcal{O}(n^2)$ states, and this closes the open problem of Sakoda–Sipser about 1nfa's vs. 2dfa's, at least in the unary case. All these results and their optimality are proved in [4].

In [8], the authors prove that $\mathcal{O}(e^{\sqrt{n \ln n}})$ is the optimal cost of simulating unary 2nfa's by 1dfa's. Moreover, just by paying a *quadratic* increase in the number of states, unary 2nfa's can always be regarded as *quasi-sweeping automata* (*qsnfa*), namely, 2nfa's having both reversals and nondeterministic choices *only* at the ends of the input. This may be seen as another step toward a "simplification" of the open question of Sakoda–Sipser.

Our work aims to give further contributions that could be helpful in shedding some light on the Sakoda–Sipser open problem. Our first result, in Section 3, improves [8], which says that, for any n-state unary 2nfa A, there exists an equivalent $\mathcal{O}(n^2)$-state qsnfa A'. Here we show that the number of states in A' can be reduced to $2n + 2$, provided that the resulting automaton is only *almost equivalent* to the original machine, that is, A and A' are allowed to disagree on a finite number of inputs. This result can be

regarded as providing a sort of *normal form* for 2nfa's, which is a two-way counterpart of the well-known Chrobak Normal Form for 1nfa's [4].

Our almost equivalent quasi-sweeping simulation becomes a useful tool for the unary version of 2nfa's vs. 2dfa's question in Section 4. Using a divide-and-conquer technique, we first show that *any n-state unary 2nfa can be simulated by an almost equivalent qsdfa with no more than* $2+(n^2+2)(2n)^{\lceil \log_2(n+1)\rceil}$ *states*. This gives a *subexponential* simulation of unary 2nfa's by fully equivalent 2dfa's, precisely:

Each n-state unary 2nfa can be simulated by an $\mathcal{O}(n^{\lceil \log_2(n+1)\rceil+3})$*-state qsdfa.*

To the best of the authors' knowledge, the previously known best simulation of unary 2nfa's by 2dfa's uses $\mathcal{O}(e^{\sqrt{n \ln n}})$ states [8]. Moreover, this result reveals a further difference between computations on general and unary alphabets. We recall, in fact, that in [12] it is proved that turning 2nfa's on general alphabets into sweeping automata has an exponential state cost.

2. Finite state automata

Here, we briefly recall some basic definitions on finite state automata. For a detailed exposition, we refer the reader to [6]. Given a set S, $|S|$ denotes its cardinality and 2^S the family of all its subsets.

A *two-way nondeterministic finite automaton* (2nfa) is defined as a quintuple $A=(Q,\ \Sigma,\delta,q_0,F)$ in which Q is the finite set of states, Σ is the finite input alphabet, $\delta: Q\times(\Sigma\cup\{\vdash,\dashv\})\to 2^{Q\times\{-1,0,+1\}}$ is the transition function, $\vdash,\dashv\notin\Sigma$ are two special symbols, called the left and the right endmarker, respectively, $q_0\in Q$ is the initial state, and $F\subseteq Q$ is the set of final states. Input is stored on the input tape surrounded by the two endmarkers, the left endmarker being in the cell no. zero. In a move, A reads an input symbol, changes its state, and moves the input head one cell forward, backward, or keeps it stationary depending on whether δ returns $+1$, -1, or 0, respectively. The machine accepts the input, if there exists a computation path from the initial state q_0 with head on the left endmarker to some final state $q\in F$. The language accepted by A, denoted by $L(A)$, consists of all input strings that are accepted. A is a two-way *deterministic* finite state automaton (2dfa) whenever $|\delta(q,\sigma)|\leqslant 1$, for any $q\in Q$ and $\sigma\in\Sigma\cup\{\vdash,\dashv\}$. In what follows, we will be particularly interested in weaker versions of 2nfa's and 2dfa's:

Definition 1.

- A *quasi-sweeping* automaton (qsnfa) is a 2nfa performing *both* input head reversals and nondeterministic choices only at the endmarkers [7].
- If, moreover, the above automaton is deterministic, we shall call it *sweeping* (qsdfa) [12].
- A (non)deterministic automaton is *one-way*, (1nfa or 1dfa, respectively), if it never moves the input head to the left.

We say that an automaton A is *almost equivalent* to an automaton A' if and only if the languages accepted by A and A' coincide, with the exception of a finite number of strings. If these two languages coincide on all strings, with no exceptions, A and A' are (*fully*) *equivalent*.

We call *unary* any automaton that works with a single letter input alphabet. In [4], it was pointed out that for unary automata 2dfa's and qsdfa's are equivalent from a descriptional point of view. In fact, it can be shown that any unary 2dfa can be substituted by an equivalent qsdfa without increasing the number of its states.

Without loss of generality, we can assume that each nondeterministic machine accepts with head on the left endmarker, not increasing the number of states:

Lemma 1. *Given a* 2*nfa* $A=(Q,\Sigma,\delta,q_0,F)$, *there exists an equivalent* 2*nfa* $A'=(Q,\Sigma,\delta',q_0,F)$ *such that*, *for each accepted input*, *there exists at least one accepting computation path reaching a final state* $q\in F$ *with the input head on the left endmarker*.

Proof. The simulation of A by A' is straightforward, until A reaches a final state $q\in F$. Then, among other possibilities, A' can move its input head to the left. Formally, the transition function δ' is defined as follows:

(1) Simulation of A: if $(p,d)\in\delta(q,\sigma)$, then $(p,d)\in\delta'(q,\sigma)$, for each $q\in Q$ and each $\sigma\in\Sigma\cup\{\vdash,\dashv\}$.
(2) Additional "new" transitions: $(q,-1)\in\delta'(q,\sigma)$, for each $q\in F$ and each $\sigma\neq\vdash$.

It is easy to see that, for each input, A' has an accepting computation path if and only if A does. Moreover, in A', each accepting path can be extended so that it reaches the left endmarker. □

The following simple lemma will be used in the paper to simplify automata by removing some states that are not actually necessary.

Lemma 2. *Given a* 2*nfa* $A=(Q,\Sigma,\delta,q_0,F)$, *there exists an equivalent* 2*nfa* $A'=(Q',\Sigma,\delta',q_0,\{q_f\})$ *not using stationary moves*, *with a possible exception in the last computation step. Here* $Q'\subseteq Q\cup\{q_f\}$ *contains a new state* $q_f\notin Q$, *the initial state* q_0, *and all other states of* Q *with the exception of those which are reachable by* A *only via stationary moves.*

Furthermore, *if* A *is quasi-sweeping*, *then* A' *is also quasi-sweeping. Similarly*, *if* A *is deterministic*, *then so is* A'.

Proof. By inspecting the image of the transition function δ, it is not difficult to find the states of A which are reachable only via stationary moves at some input symbols. Thus, the new state set Q' can be easily computed. The transition function δ' of A' is defined by considering the following transitions:

(1) *Transitions on a symbol* $\sigma\in\Sigma\cup\{\vdash,\dashv\}$: these are defined by replacing sequences of stationary transitions followed by one move to the left or right with one equivalent transition step. To this end, consider the relation $R_\sigma^+\subseteq(Q'-\{q_f\})\times(Q'-\{q_f\})$, consisting of all pairs (q,p) of states such that the automaton A starting from the state q with the input head scanning the symbol σ, after a sequence of

stationary moves followed by a single move to the right, can reach the state p. For each pair $(q,p)\in R_\sigma^+$, we define the transition $(p,+1)\in\delta'(q,\sigma)$. In a similar way, using R_σ^-, we can introduce transitions moving the input head one cell to the left.

(2) *Transitions to the final state*: it can happen that the automaton A can reach a final state by a sequence of stationary moves. This situation is resolved in A' as follows: redefine $\delta'(q,\sigma)$ to $\delta'(q,\sigma)=\{(q_f,0)\}$, for each $q\in Q'-\{q_f\}$ and each $\sigma\in\Sigma\cup\{\vdash,\dashv\}$, such that the automaton A, with the input head scanning the symbol σ, can get from the state q to some final state using only a (possibly empty) sequence of stationary moves. This includes the case of $q\in F$.

It is not difficult to verify that the languages accepted by the automata A and A' coincide, that A' is quasi-sweeping if A is quasi-sweeping, and that the above transformation preserves determinism. Moreover, if A always accepts with the head parked at the left endmarker, then so does A', in the state q_f. □

3. Linear, almost equivalent, quasi-sweeping simulation

In this section, we show how to get, from a unary n-state 2nfa, an almost equivalent qsnfa with no more than $2n+2$ states. From now on, we will always refer to a *unary 2nfa A with n states, accepting with the input head on the left endmarker* (see Lemma 1).

First of all, we recall some results, mainly from [5,8], concerning the "form" of accepting computations of A. By a *loop of length* ℓ, we mean a computation path of A beginning in a state p with the input head at a position i, ending in the same state with the input head at the position $i+\ell$, and not visiting the endmarkers in the meantime.

Consider an accepting computation of A on input 1^m. Let $r_0,r_1,\ldots,r_p$ be the sequence of all states in which the input head scans either of the endmarkers. Note that $r_0=q_0$ and $r_p\in F$. For $1\leqslant j\leqslant p$, the following two possibilities arise:

- In both r_{j-1} and r_j, the input head scans the same endmarker. This segment of computation is called a U-*turn*.
- In r_{j-1} the input head scans one of the two endmarkers, while in r_j it scans the other. This segment of computation is called a (*left-to-right or right-to-left*) *traversal*. (Notice that, within a traversal, the endmarkers are never touched.)

Lemma 3 (Geffert [5], Mereghetti and Pighizzini [8]). *Given two states* q_1,q_2 *of* A, *and an input* 1^m, *with* $m>n$:

(i) *for each U-turn from* q_1 *to* q_2, *there is another U-turn from* q_1 *to* q_2 *in which the input head is never moved farther than* n^2 *cells from the corresponding endmarker*;

(ii) *for each traversal from* q_1 *to* q_2, *there is a traversal from* q_1 *to* q_2 *where* A:
  (a) *having traversed the starting endmarker and* s_1 *cells*,
  (b) *gets into a loop* (*called* dominant loop) *of length* ℓ, *which starts from a state* p *and is repeated* λ *times*,

(c) *then it traverses the remaining* s_2 *input squares, and finally reaches the other endmarker,*

for some p, *and* λ, s_1, s_2, ℓ *satisfying* $0 \leqslant \lambda$, $1 \leqslant |\ell| \leqslant n$, *and* $s_1 + s_2 \leqslant 3n^2$.

To study possible loop lengths, it is useful to consider the weighted digraph $\mathscr{A}$ with edges representing the transition diagram of our 2nfa A after removing transitions on the endmarkers, and in which we set weights $+1$, -1, or 0 to arcs depending on whether they represent transitions where the input head is moved right, left, or kept stationary, respectively. It is straightforward that *any cycle of weight* ℓ *in* $\mathscr{A}$ *represents a computation loop of length* ℓ *in the automaton* A, taking place "sufficiently far" from either endmarker, so that neither endmarker can be visited along the loop. Let us partition the digraph $\mathscr{A}$ into strongly connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_r$. Let $\ell_i > 0$ denote the greatest common divisor of the absolute values of cycle weights in the component $\mathscr{C}_i$, for $1 \leqslant i \leqslant r$. It is easy to see that

$$\ell_1 + \ell_2 + \cdots + \ell_r \leqslant n,$$

since $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_r$ use pairwise disjoint sets of states of the machine A.

By Lemma 3(ii), it is possible to prove that input traversals can be both expanded and compressed. More precisely, it can be shown:

Lemma 4 (Mereghetti and Pighizzini [8]). *If there exists a traversal on the input* 1^m, $m > n$, *from a state* q_1 *to a state* q_2, *whose dominant loop uses states belonging to the component* $\mathscr{C}_i$, *then there also exists another traversal from* q_1 *to* q_2 *on* $1^{m+\mu\ell_i}$, *for any integer* $\mu > (5n^2 - m)/\ell_i$.

Note that for $m > 5n^2$ we can also use *negative* values of μ, i.e., the input can be compressed.

Using these results, a qsnfa A' with $\mathcal{O}(n^2)$ states, equivalent to a given 2nfa A, is built in [8]. Without going into details, we just recall that the simulation performed by A' basically consists of two phases: in the first phase, A' checks membership in $L(A)$ for strings of length not exceeding $5n^2$. This phase clearly takes $\mathcal{O}(n^2)$ states. Longer strings are handled in a second phase, using another $\mathcal{O}(n^2)$ states.

We are now going to show how to decrease, to $2n+2$ states, the cost of the second phase so as to obtain a linear almost equivalent simulation.

The state set Q' of A' will be the union of the state set Q of A with two new sets of states Q^+ and Q^-. The set Q will be used to simulate A on the endmarkers only. In particular, U-turns will be precomputed and simulated with stationary moves. The set Q^+ (resp. Q^-) will be used to simulate left-to-right (resp. right-to-left) traversals. For instance, a cycle of ℓ_i states belonging to Q^+ will be used to simulate "dominant loops" involving states of the strongly connected component $\mathscr{C}_i$ in a left-to-right traversal. More precisely, for $1 \leqslant i \leqslant r$, we define $Q_i^+ = \{q_{i,0}^+, q_{i,1}^+, \ldots, q_{i,(\ell_i-1)}^+\}$ and $Q_i^- = \{q_{i,0}^-, q_{i,1}^-, \ldots, q_{i,(\ell_i-1)}^-\}$, with the following deterministic transitions:

$$\delta'(q_{i,k}^+, 1) = \{(q_{i,(k+1) \bmod \ell_i}^+, +1)\} \quad \text{and} \quad \delta'(q_{i,k}^-, 1) = \{(q_{i,(k+1) \bmod \ell_i}^-, -1)\}.$$

Thus, the state set of A' is $Q' = Q \cup Q^+ \cup Q^-$, where $Q^+ = \bigcup_{i=1}^{r} Q_i^+$, and $Q^- = \bigcup_{i=1}^{r} Q_i^-$. Since $\ell_1 + \ell_2 + \cdots + \ell_r \leqslant n$, it is obvious that there are at most $3n$ states in Q'.

Let us now see how to map the states of A into the states of A'. We first explain how this mapping works on states involved in left-to-right traversals. Its extension to encompass traversals in the opposite way can be argued easily. Before defining $\varphi : Q \to Q^+$, we need some notation. given two states $p, q \in Q$, we write $p \rhd^{+x} q$ if there exists a computation path of A which starts from p with the input head at a position d, ends in q at the position $d + x$, and does not visit the endmarkers. By Lemma 3(i), it can be shown that such a path does not depend on the input head position d, provided that both d and $d + x$ are at least n^2 positions away from either endmarker (for details, see [5]). A component $\mathscr{C}_i$ is said to be *positive* (resp. *negative*), if it contains at least one cycle of positive (negative) weight. Note that a component can be, at the same time, positive and negative. Furthermore, all the states used in the dominant loop of a left-to-right traversal (see Lemma 3) clearly belong to the same positive component. Given a *positive* component $\mathscr{C}_i$, we designate some state $q_{(i)} \in \mathscr{C}_i$ as the *center* of the component and, for any state $p \in \mathscr{C}_i$, we define

$$\kappa(p) = \min\{x \in \mathbf{N}\colon\ q_{(i)} \rhd^{+x} p\} \quad \text{and} \quad \varphi(p) = q^+_{i,\kappa(p) \bmod \ell_i}.$$

This mapping is well defined since $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_r$ are disjoint sets. Note also that $\varphi(q_{(i)}) = q^+_{i,0}$, and that $q_{(i)} \rhd^{+\kappa(p)} p$, for each $p \in \mathscr{C}_i$. The mapping φ defines a partition of the set of states in $\mathscr{C}_i$; states in the same class have the same "distance" modulo ℓ_i from the center and, for sufficiently large inputs, they can be considered equivalent:

Lemma 5. *Let $q, p \in Q$ be in the same positive component $\mathscr{C}_i$. Then $q \rhd^{+m} p$ in A if and only if $\varphi(q) \rhd^{+m} \varphi(p)$ in A', for each $m \geqslant 2n^2 + n$.*

Proof. We first show that, for each two states q', p' within the same component $\mathscr{C}_i$, and for each two integers x_1, x_2, such that $q' \rhd^{+x_1} p'$ and $q' \rhd^{+x_2} p'$, we have $x_1 \bmod \ell_i = x_2 \bmod \ell_i$. The strongly connected component $\mathscr{C}_i$ must have, for some integer h, a path $p' \rhd^{+h} q'$, and hence also two loops, namely $q' \rhd^{+(x_1+h)} q'$ and $q' \rhd^{+(x_2+h)} q'$. But then both $x_1 + h$ and $x_2 + h$ must be some integer multiples of ℓ_i, i.e., $(x_1 + h) \bmod \ell_i = (x_2 + h) \bmod \ell_i = 0$, and hence $x_1 \bmod \ell_i = x_2 \bmod \ell_i$.

Thus, if we replace the path $q_{(i)} \rhd^{+\kappa(p)} p$ by a path $q_{(i)} \rhd^{+\kappa(q)} q \rhd^{+m} p$, we get $\kappa(p) \bmod \ell_i = (\kappa(q) + m) \bmod \ell_i$. Therefore, $q \rhd^{+m} p$ implies that $m \bmod \ell_i = (\kappa(p) - \kappa(q)) \bmod \ell_i$.

Conversely, let $m \bmod \ell_i = (\kappa(p) - \kappa(q)) \bmod \ell_i$, for some $m \geqslant 2n^2 + n$. Since $\mathscr{C}_i$ is strongly connected, we can find a path $q \rhd^{+m'} p$, for some $m' < n$. By the argument above, m' must satisfy $m' \bmod \ell_i = (\kappa(p) - \kappa(q)) \bmod \ell_i = m \bmod \ell_i$. Thus, to get a path $q \rhd^{+m} p$ of weight m, it is sufficient to insert a suitable number of cycles beginning and ending in q, into the path $q \rhd^{+m'} p$. This is possible, since the set of integers $x > 2n^2$ such that A has a computation path starting from the state q of the component $\mathscr{C}_i$ with the input head at a position d, ending at the position $d + x$ in the same state q, and not visiting the endmarkers in the meantime, coincides with the set of integer multiples of

ℓ_i greater than $2n^2$. (For details, see [8, Lemma 3.4].) Thus, $q \rhd^{+m} p$ if and only if $m \bmod \ell_i = (\kappa(p) - \kappa(q)) \bmod \ell_i$, for each $m \geqslant 2n^2 + n$.

On the other hand, from the definition of δ' and φ presented above, the machine A' has a path $\varphi(q) = q^+_{i,\kappa(q) \bmod \ell_i} \rhd^{+m} q^+_{i,\kappa(p) \bmod \ell_i} = \varphi(p)$ if and only if $m \bmod \ell_i = (\kappa(p) - \kappa(q)) \bmod \ell_i$, for each $m > 0$. (The definition of δ' has not been completed yet, however, the transitions we are going to introduce in the future do not invalidate the above property for states in Q^+; they will only describe what happens when the input head scans an endmarker). □

Let us now define the behavior of A' on the endmarkers. We recall that, as previously observed, a computation of A can be decomposed into U-turns and traversals. Note that, by Lemma 3(i), U-turns do not depend on the input length m, if $m > n^2$. So, to simulate U-turns, we set, for each $p, q \in Q$:

- $(q, 0) \in \delta'(p, \vdash)$ (similarly, $(q, 0) \in \delta'(p, \dashv)$) if and only if there exists a U-turn on the left (right) endmarker starting in the state p and ending in q.

To simulate traversals, we must introduce further moves on the endmarkers. In what follows, let

$$M = \rho \cdot \left\lceil \frac{3n^2 + 1}{\rho} \right\rceil,$$

where ρ is the least common multiple of $\ell_1, \ell_2, \ldots, \ell_r$.

Again, we first concentrate on left-to-right traversals. Using the form recalled in Lemma 3(ii), we substitute such a traversal with a deterministic computation executing one of the simple cycles we have introduced. The initial and the final parts of the traversal are precomputed. We define such moves by considering a very large input, of length $2M$. It turns out that once we can correctly simulate a traversal of A on the input 1^{2M}, we can correctly simulate any traversal on any sufficiently large input, of length $m > 5n^2$.

Suppose now that the input length is $2M$. After reading the left endmarker and the first M input symbols, a state p is reached. It is easy to see that p belongs to the dominant loop, and hence also to some positive component $\mathscr{C}_i$, since $M > 3n^2$, but the initial and final segments of a traversal described in Lemma 3(ii) are of lengths $s_1 + s_2 \leqslant 3n^2$. We want the simulating automaton A' to reach the state $\varphi(p)$ after reading this portion of the input. To this aim, since M is an integer multiple of ℓ_i, it suffices to start the traversal directly in the state $\varphi(p)$. So, we simulate this segment of computation by a single move on the left endmarker. Similarly, we want the simulating automaton A', starting from $\varphi(p)$ placed M cells away from the right endmarker, to reach the same state as A does, when started from p and having traversed the final M cells of the input. Again, a single move suffices, this time from $\varphi(p)$ at the right endmarker, since ℓ_i divides M. Thus, for $q \in Q$ and $\tilde{p} \in Q^+$, we set:

- $(\tilde{p}, 1) \in \delta'(q, \vdash)$ if and only if there exists $r \in Q$ and $p \in \varphi^{-1}(\tilde{p})$ such that $(r, 1) \in \delta(q, \vdash)$ and $r \rhd^{+M} p$ in A;
- $(q, 0) \in \delta'(\tilde{p}, \dashv)$ if and only if there exists $p \in \varphi^{-1}(\tilde{p})$ with $p \rhd^{+M} q$ in A.

Similar moves can be used to simulate right-to-left traversals. These are defined for components $\mathscr{C}_i$ that are *negative*, i.e., having at least one cycle of negative weight.

Here we use dual functions

$$\hat{\kappa}(p) = \min\{x \in \mathbf{N}:\ q_{(i)} \rhd^{-x} p\}, \quad \text{and} \quad \hat{\varphi}(p) = q^{-}_{i,\hat{\kappa}(p) \bmod \ell_i}.$$

Jumping to and fro $\tilde{p} \in Q^-$ is defined symmetrically to that of $\tilde{p} \in Q^+$:

- $(\tilde{p}, -1) \in \delta'(q, \dashv)$ if and only if there exists $r \in Q$ and $p \in \hat{\varphi}^{-1}(\tilde{p})$ such that $(r, -1) \in \delta(q, \dashv)$ and $r \rhd^{-M} p$ in A;
- $(q, 0) \in \delta'(\tilde{p}, \vdash)$ if and only if there exists $p \in \hat{\varphi}^{-1}(\tilde{p})$ with $p \rhd^{-M} q$ in A.

At this point, our simulating quasi-sweeping automaton is completely defined as $A' = (Q', \{1\}, \delta', q_0, F)$, where Q' and δ' have been presented so far, while q_0 and F are, respectively, the initial state and the set of final states of A.

Theorem 1. *Let $m > 5n^2$ and $q_1, q_2 \in Q$. There exists a left-to-right traversal of A on the input 1^m from q_1 to q_2 if and only if there exists a left-to-right traversal of A' from q_1 which is followed by a stationary move to reach q_2.*

An analogous statement holds for traversals from right to left.

Proof. We prove the equivalence for left-to-right traversals only. Its validity for the symmetric case is straightforward.

First, by Lemma 3(ii), each traversal of 1^m, for $m > 5n^2$, has a dominant loop with states belonging to some positive component $\mathscr{C}_i$. Note that $2M$ is an integer multiple of ℓ_j, for each $j = 1, \ldots, r$. That is, $2M = \mu_i \ell_i$, for some $\mu_i > 0 > (5n^2 - m)/\ell_i$. Thus, by Lemma 4, there exists a left-to-right traversal of A from q_1 to q_2 on the input $1^{m+\mu_i \ell_i} = 1^{m+2M}$. Conversely, a left-to-right traversal on the input 1^{m+2M} implies the existence of a dominant loop in a component $\mathscr{C}_i$, for some i. Using Lemma 4, this time with $\mu_i = -2M/\ell_i > (5n^2 - m - 2M)/\ell_i$, we get a traversal of A from q_1 to q_2 on the input $1^{m+2M+\mu_i \ell_i} = 1^m$.

In conclusion, there exists a left-to-right traversal of A on the input 1^m from q_1 to q_2 if and only if the same holds for the input 1^{m+2M}. So, for A, consider the input 1^{m+2M} instead of 1^m.

Only if part: We subdivide a left-to-right traversal of A on 1^{m+2M} into the following three phases:

Phase 1: A leaves the left endmarker from the state q_1 and enters a state $r \in Q$, in a single step. From r, it starts consuming the next M input symbols till it reaches a state p.

Phase 2: From p, it moves m more symbols to the right, where it reaches a state q.

Phase 3: From q, the last M symbols are consumed, and the right endmarker is finally reached in the state q_2.

By Lemma 3(ii), we may assume, without loss of generality, that the above traversal iterates a dominant loop. But then, since $M > 3n^2$, both p and q belong to the same positive component $\mathscr{C}_i$, for some i.

Let us see how the above phases are reproduced by A' on the input 1^m. By definition of δ', Phase 1 implies that $(\varphi(p), 1) \in \delta'(q_1, \vdash)$, i.e., A' reaches the state $\varphi(p)$ by consuming the left endmarker. Next, by Phase 2 and Lemma 5, we get that $\varphi(p) \rhd^{+m} \varphi(q)$, i.e., A' reaches the right endmarker in the state $\varphi(q)$. Finally, Phase 3 and the definition

of δ' ensure that $(q_2,0)\in\delta'(\varphi(q),\dashv)$, i.e., A' enters the state q_2 with a stationary move on the right endmarker.

If part: A left-to-right traversal of A' on 1^m from q_1 to q_2 begins with a first move that takes A' from q_1 to some state $\tilde{p}\in Q^+$, with the input head on the first '1'. By definition of δ', we know that there exists $p\in\varphi^{-1}(\tilde{p})$ that is reached by A from q_1 after consuming the left endmarker and the first M symbols of the input. Since $\tilde{p}\in Q^+$, A' enters a deterministic loop then, that takes it to some state $\tilde{q}\in Q^+$ when the right endmarker is reached. The loop consumes all m symbols of the input, hence, $\tilde{p}\rhd^{+m}\tilde{q}$ in A'. Finally, A' has a stationary move on the right endmarker, that takes it from $\tilde{q}\in Q^+$ to $q_2\in Q$. By definition of δ', there must exist some $q\in\varphi^{-1}(\tilde{q})$ such that some computation path of A, starting in q, reaches the right endmarker in q_2, having moved M input tape symbols to the right. Note also that φ maps p and q into the states $\tilde{p}$ and $\tilde{q}$, respectively, of the same loop in Q_i^+, for some $1\leqslant i\leqslant r$, and hence both p and q belong to the same positive component $\mathscr{C}_i$. To get a left-to-right traversal of A from q_1 to q_2 on the input 1^{m+2M}, it only remains to show that A can get from p to q by traversing m tape cells to the right. This is easy, since $\varphi(p)=\tilde{p}\rhd^{+m}\tilde{q}=\varphi(q)$ in A', and hence, by Lemma 5, $p\rhd^{+m}q$ in A. □

We are now ready to prove.

Lemma 6. *For each n-state unary 2nfa A, there exists an almost equivalent qsnfa A' with no more than $3n$ states. Moreover, $L(A)$ and $L(A')$ coincide on strings of length greater than $5n^2$.*

Proof. Recall that, by Lemma 1, we may assume, without loss of generality, that A starts and accepts with the input head on the left endmarker. This machine is replaced by A' described above. Note that A' uses the same set of final states $F\subseteq Q\subset Q'$ and that a state $q\in Q$ can be reached by A' only when an endmarker is scanned. Thus, it suffices to prove that A and A' get to the same states of Q whenever the input head scans either of the endmarkers. The argument for this is a straightforward induction on the number of times the input head visits the endmarkers, using Theorem 1, including its right-to-left version, and the fact that U-turns are precomputed in A'. □

At this point, Lemma 2 enables us to derive, from A', a qsnfa A'' which does not use the states of the original 2nfa A. This leads us to the main result of this section.

Theorem 2. *For each n-state unary 2nfa A, there exists an almost equivalent qsnfa A'' with no more than $2n+2$ states. Moreover, $L(A)$ and $L(A'')$ coincide on strings of length greater than $5n^2$.*

Proof. The state set of the automaton A' of Lemma 6 is given by $Q\cup Q^+\cup Q^-$. In particular, the states in Q are reachable only via stationary moves at the endmarkers. Hence, in the light of Lemma 2, we can get an equivalent 2nfa by removing them, with the only exception of the initial state q_0, and by adding a new final state. Thus, the total number of states of A'' turns out to be bounded by $2n+2$. □

We end this section with some observations:

- In [4], a unary n-state one-way nondeterministic finite automaton is turned into an equivalent 1nfa consisting of an initial path of $\mathcal{O}(n^2)$ states ending in a state where a nondeterministic choice is taken. Such a choice leads into one among a certain number of disjoint cycles, and the rest of the computation is deterministic. The total number of states included in the cycles does not exceed n. This structure is usually known as Chrobak Normal Form for unary 1nfa's. In our framework, this result can be reformulated by saying that there exists an *almost equivalent* n-state 1nfa in which the only nondeterministic decision is taken at the beginning of the computation. From this point of view, Theorem 2 can be regarded as an extension of this result to the two-way machines, and might suggest a sort of *normal form for 2nfa's*.
- If the language accepted by the given 2nfa A is cyclic—i.e., for some $\lambda > 0$ and for each $m \geqslant 0$, $1^m \in L(A)$ if and only if $1^{m+\lambda} \in L(A)$—the automaton A'' yielded by our construction is fully equivalent to A.
- The automaton A'' has been constructed by application of Lemma 1, 6, and Theorem 2 (actually using Lemma 2), in that order. As a result, A'' has at least one accepting path ending with the head parked on the left endmarker, in the unique state q_f, for each accepted input. Another interesting property of A'' is the following symmetry; if we reverse the orientation of all edges representing the transition diagram and swap the roles of q_0 and q_f, we get a new automaton, that is quasi-sweeping again, and recognizes the same language. In other words, away from the endmarkers, each state of A'' has not only a unique successor (a consequence of being quasi-sweeping), but also a unique predecessor as well.

4. Subexponential deterministic simulation

In this section, we show how to simulate an n-state unary 2nfa by a qsdfa with only $\mathcal{O}(n^{\lceil \log_2(n+1) \rceil + 3})$ states, thus improving previous bounds in the literature. To this end, we shall use the almost equivalent quasi-sweeping automaton presented in the previous section. From now on, unless otherwise stated, we will always refer to the *unary qsnfa A'', constructed in Theorem 2*.

Recall that A'' accepts by entering a unique state q_f with the input head scanning the left endmarker. The core of our simulation technique is the implementation of the predicate *reachable* which is defined as follows. Fix an input length m, the states q, p, and an integer $k \geqslant 1$. Then *reachable*(q, p, k) is true if and only if there exists a computation path of A'' which starts and ends with the input head scanning the left endmarker in the state q and p, respectively, and visits that endmarker at most k times (excluding q, including p). It is easy to see that if *reachable*(q, p, k) holds true, then there exists a witness computation path where the states encountered when the input head scans the left endmarker are all different. By closely observing the structure of A'', as outlined in the proof of Theorem 2, we can notice that the only states that it can use when the input head is on the left endmarker are the initial state q_0, the final state q_f, and the states in Q^-. Since $|Q^-| \leqslant n$,

this implies that 1^m *is accepted if and only if* $reachable(q_0, q_f, n+1)$ *holds true.*

To evaluate the predicate *reachable*, it is useful to concentrate first on computing its "simplest" case $reach1(q,p) = reachable(q,p,1)$, for any q and p.

Lemma 7. *For each fixed pair of states* $q, p \in Q^- \cup \{q_0, q_f\}$, *the value of* $reach1(q,p)$ *can be computed by a qsdfa* $A_{q,p}$ *with at most* n^2+3 *states.*

Proof. The predicate $reach1(q,p)$ holds true if and only if either (i) $q=p$, or (ii) $(p,0) \in \delta(q,\vdash)$, which, by Theorem 2 (see also Lemma 2), can happen only if $p=q_f$, or (iii) there is a path from q to p which consists of a left-to-right traversal of the input, using a deterministic loop of length ℓ_i in Q^+, corresponding to some positive component $\mathscr{C}_i$, followed by a traversal in the opposite way, using a loop of length ℓ_j in Q^-, for some negative component $\mathscr{C}_j$. Nondeterministic choices can be used only when the input head is leaving one of the endmarkers.

If one of the Cases (i) or (ii) holds true, $A_{q,p}$ halts immediately in the exit state q_{yes}. Case (iii) can be tested by trying all possible choices of pairs $(\mathscr{C}_i, \mathscr{C}_j)$, where $\mathscr{C}_i$ is a positive component of the original machine, while $\mathscr{C}_j$ is a negative component. The machine $A_{q,p}$ starts by moving one cell to the right from the left endmarker, to a state corresponding to the pair $(\mathscr{C}_i, \mathscr{C}_j)$ of the first positive and the first negative, respectively, components.

For each given pair of components $(\mathscr{C}_i, \mathscr{C}_j)$, the machine uses two separate counters x, y, initially set to zero. Starting from the first cell, $A_{q,p}$ traverses the input 1^m from left to right, counting simultaneously $x = m \bmod \ell_i$ and $y = m \bmod \ell_j$. For a fixed pair $(\mathscr{C}_i, \mathscr{C}_j)$, a deterministic loop consisting of $\ell_i \cdot \ell_j$ states is sufficient. When the input head reaches the right endmarker, $A_{q,p}$ checks whether the following conditions hold true:

(1) there exist $a \in \{0, \ldots, \ell_i - 1\}$ and $b \in \{0, \ldots, \ell_j - 1\}$, such that
(2) A'' can get from the state q to the state $q^+_{i,a} \in Q^+$, by a single move to the right,
(3) this move is followed by a deterministic loop that ends in the state $q^+_{i,c}$ at the right endmarker, for $c = (a+x) \bmod \ell_i$,
(4) from $q^+_{i,c}$ at the right endmarker, A'' can get, by a single move to the left, to the state $q^-_{j,b} \in Q^-$,
(5) this move is followed by a deterministic loop that ends in the state $q^-_{j,d}$ at the left endmarker, for $d = (b+y) \bmod \ell_j$, satisfying $q^-_{j,d} = p$.

It should be clear that, having computed $x = m \bmod \ell_i$ and $y = m \bmod \ell_j$, $A_{q,p}$ has enough information to verify conditions (1)–(5), in a single transition at the right endmarker. If conditions (1)–(5) turn out to be true, $A_{q,p}$ enters the exit state q_{yes}. Otherwise, moving one cell to the left from the right endmarker, it selects another pair of components $(\mathscr{C}_i, \mathscr{C}_j)$ and counts the length of the input modulo new values of ℓ_i and ℓ_j, traversing this time the input from right to left. This way, one after another, moving alternately from the left and right endmarkers, all possible pairs $(\mathscr{C}_i, \mathscr{C}_j)$ are tested. When all combinations of $(\mathscr{C}_i, \mathscr{C}_j)$ have been exhausted, $A_{q,p}$ enters the exit state q_{no}. Finally, since we want $A_{q,p}$ to halt always at the left endmarker, $A_{q,p}$ may need to traverse the

input in one of the exit states, using transitions $(r,-1)\in\delta(r,\sigma)$, for $r\in\{q_{\rm yes},q_{\rm no}\}$ and $\sigma\in\{1,\dashv\}$.

For each pair of components $(\mathscr{C}_i,\mathscr{C}_j)$, the machine uses a separate deterministic loop consisting of $\ell_i\cdot\ell_j$ states. In addition, it has one initial plus two exit states. Summing up, the number of states is bounded by $3+\sum_{i=1}^{r}\sum_{j=1}^{r}\ell_i\cdot\ell_j\leqslant 3+n^2$, using the fact that $\ell_1+\ell_2+\cdots+\ell_r\leqslant n$. □

Now, we use *reach1* as a subroutine to compute the predicate *reachable*:

```
function reachable(q, p, k)
if k = 1 then return reach1(q, p)
else begin
  for each state r ∈ Q⁻ do
      if reachable(q, r, ⌈k/2⌉) then
        if reachable(r, p, ⌈k/2⌉) then
          return TRUE
  return FALSE
end
```

Such a function can be implemented by using a pushdown store in which, at each position, a pair of states and an integer in the range $1\ldots n+1$, corresponding to one activation of the function, are kept. The maximal pushdown height is $\lceil\log_2(n+1)\rceil$, not counting the activation of the "main program", i.e., the predicate $reachable(q_0,q_{\rm f},n+1)$. Note that, unlike in a classical divide-and-conquer technique, the problem of size k is not divided into two subproblems of sizes $\lfloor k/2\rfloor$ and $\lceil k/2\rceil$, but, rather, both the outer and inner **if** statements use the same parameter $\lceil k/2\rceil$. This ensures that, whenever a bottom level of the recursion is reached, the pushdown is of the same height.

To reduce the number of possible configurations, we utilize not only the element at the top position of the pushdown, but all information that is currently stored.[2] When a state $r\in Q^-$ is considered in the **for** loop of the function, the call $reachable(q,r,\lceil k/2\rceil)$ is simulated by adding the state r on top of the stack, while the call $reachable(r,p,\lceil k/2\rceil)$ is indicated by replacing r with a marked copy $\hat{r}$ on top of the stack.

We now informally explain how, from this stack, it is possible to recover the original pushdown store. For the sake of simplicity, we give an example of the computation of *reachable* for a qsnfa A'' with $n=56$ states in Q^-. (With some technicalities, the argument can easily be extended to any n.) Suppose that the current stack contents, from the bottom to the top, is $\hat{q}_3\,q_2\,q_5\,\hat{q}_7$. By also representing, implicitly, $q_{\rm f}$ and $\hat{q}_0$ at the bottom, we have the following situation:

STACK $\Longrightarrow$

position	−1	0	1	2	3	4
content	$q_{\rm f}$	$\hat{q}_0$	$\hat{q}_3$	q_2	q_5	$\hat{q}_7$

[2] Such a device, which is less restrictive than a pushdown, is sometimes denoted by the term stack in the literature (see, e.g., [6]).

Each stack position s containing a marked state corresponds to an activation of *reachable* in the inner **if** statement. The other state of this activation is the first nonmarked state to the left of position s. In a similar way, we can recover an activation for a nonmarked state in the stack, i.e., for the outer **if** statement, by searching for the first marked state in the stack placed to the left of position s. The third parameter of the activation at position s is k_s, the limit for the number of visits at the left endmarker. This parameter can be easily computed as follows. For $s=0$, the activation of the "main program", we always have $k_0 = n+1$. Further, for $s>0$, we have $k_s = \lceil k_{s-1}/2 \rceil$. In the above example, the corresponding sequence of current activations of *reachable* is

$$(q_0, q_f, 57),\ (q_3, q_f, 29),\ (q_3, q_2, 15),\ (q_3, q_5, 8), (q_7, q_5, 4).$$

The computation of *reachable* $(q_0, q_f, n+1)$ can be implemented by a qsdfa B which keeps in its state a stack configuration corresponding to the sequence of current activations of *reachable*. At the bottom level of the recursion, *reachable*$(q, p, 1)$ is verified by a subautomaton $A_{q,p}$, described in Lemma 7.

Since the pushdown height is bounded by $\lceil \log_2(n+1) \rceil$, and each stack position can contain one of $2n$ possible states of $Q^- \times \{\text{marked}, \text{nonmarked}\}$ (the states q_0 or q_f are never stored in the stack), the number of different stack configurations is at most $\sum_{i=0}^{\lceil \log_2(n+1) \rceil} (2n)^i$. By also considering the cost n^2+3 of implementing the function *reach1* of Lemma 7, we get that the resulting qsdfa B uses $(n^2+3)\sum_{i=0}^{\lceil \log_2(n+1) \rceil} (2n)^i$ states.

However, stack configurations not corresponding to the bottom level of the recursive activations of *reachable* (the stack is not full) represent, in B, the states that are reachable only via stationary moves at the left endmarker. The next-state function of B either (i) adds one state on top of the stack, (ii) replaces one state on top by another, or (iii) removes one state from top, not moving the input head. The subautomaton verifying *reach1* by scanning the input is activated only if the stack is full. Thus, by application of Lemma 2, we get a new automaton B', in which all states correspond to stack configurations at the bottom level of the recursion, with two exceptions, the original initial state, preserved by Lemma 2, corresponding to the initial contents of the stack, and a new final state q_f. Starting with the head on the left endmarker, B' simulates the execution of the function *reach1*, and then, depending on the outcome of this simulation, it selects another full-stack configuration. Recall that all full-stack configurations are of the same height. Thus, the number of full-stack configurations, corresponding to the bottom level of the recursion, is bounded[3] by $(2n)^{\lceil \log_2(n+1) \rceil}$.

Further, each full-stack configuration unambiguously determines the parameters for the function *reach1*(q, p), and hence also the subautomaton $A_{q,p}$ of Lemma 7 to be used. A careful observation reveals that Lemma 2 reduces the cost n^2+3 for $A_{q,p}$ by one state, since, in B, the initial state of $A_{q,p}$, for any given full-stack configuration,

[3] Note that for any accepting computation of A'', there actually exists an accepting computation which does not visit the left endmarker in the same state twice. Thus, we can implement the **for** loop of the function *reachable* by considering only states that are not already in the stack. This enables us to lower the total number of stack configurations to $2^{\lceil \log_2(n+1) \rceil} n(n-1)\cdot\ldots\cdot(n-\lceil \log_2(n+1) \rceil + 1)$.

is reachable only via sequences of stationary moves from some exit states q_{yes} or q_{no} of another subautomata $A_{q',p'}$, associated with some other full-stack configurations (or from the initial state of B). Thus, the initial states of subautomata implementing *reach1* are removed by application of Lemma 2. This gives:

Theorem 3. *For each n-state unary 2nfa A, there exists an almost equivalent qsdfa B' with no more than $2+(n^2+2)\cdot(2n)^{\lceil \log_2(n+1)\rceil}$ states. Moreover, $L(A)$ and $L(B')$ coincide on strings of length greater than $5n^2$.*

Theorem 4. *For each n-state unary 2nfa, there exists an equivalent qsdfa with $\mathcal{O}(n^{\lceil \log_2(n+1)\rceil+3})$ states.*

Proof. The final qsdfa B'' equivalent to A works as follows. In a first phase, it simulates an automaton with $5n^2+1$ states accepting the strings in $L(A)$ of length not exceeding $5n^2$. Then, if the input length exceeds $5n^2$, it simulates B'. It is easy to bound the total number of states in B'' by $\mathcal{O}(n^{\lceil \log_2(n+1)\rceil+3})$. □

We remark that, as far as the authors know, the subexponential simulation cost contained in Theorem 4 represents an improvement of the best unary simulation of 2nfa's by 2dfa's known in the literature which used $\mathcal{O}(e^{\sqrt{n \ln n}})$ states [8].

References

[1] P. Berman, A note on sweeping automata, in: J.W. de Bakker, J. van Leeuwen (Eds.), Proc. 7th Internat. Colloq. on Aut. Lang. Progr., Lecture Notes in Computer Science, Vol. 85, Springer, Berlin, 1980, pp. 91–97.

[2] P. Berman, A. Lingas, On the complexity of regular languages in terms of finite automata, Tech. Rep. 304, Polish Academy of Sciences, 1977.

[3] J.-C. Birget, Two-way automaton computations, Theoret. Inform. Appl. 24 (1990) 47–66.

[4] M. Chrobak, Finite automata and unary languages, Theoret. Comput. Sci. 47 (1986) 149–158.

[5] V. Geffert, Nondeterministic computations in sublogarithmic space and space constructibility, SIAM J. Comput. 20 (1991) 484–498.

[6] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.

[7] C. Mereghetti, G. Pighizzini, Two-way automata simulations and unary languages, J. Automat. Lang. Combin. 5 (2000) 287–300.

[8] C. Mereghetti, G. Pighizzini, Optimal simulations between unary automata, SIAM J. Comput. 30 (2001) 1976–1992.

[9] S. Micali, Two-way deterministic finite automata are exponentially more succinct than sweeping automata, Inform. Process. Lett. 12 (1981) 103–105.

[10] N. Piterman, M.Y. Vardi, From bidirectionality to alternation, in: J. Sgall, A. Pultr, P. Kolman (Eds.), Proc. 26th Math. Found. Computer Science, Lecture Notes in Computer Science, Vol. 2136, Springer, Berlin, 2001, pp. 598–610.

[11] W. Sakoda, M. Sipser, Nondeterminism and the size of two-way finite automata, in: Proc. 10th ACM Symp. on Theory of Computing, 1978, pp. 275–286.

[12] M. Sipser, Lower bounds on the size of sweeping automata, J. Comput. System Sci. 21 (1980) 195–202.

[13] M.Y. Vardi, A note on the reduction of two-way automata to one-way automata, Inform. Process. Lett. 30 (1989) 261–264.